X-Stack: Auto-tuning for Performance and Productivity on Extreme-scale Computations

Samuel Williams, Leonid Oliker, John Gilbert, Stephane Ethier, Kamesh Madduri, Aydin Buluc

Abstract

In the 1990's, advances in CMOS technology, cooling, and superscalar out-of-order architectures allowed for an unprecedented acceleration in computing power beyond Moore's law. Unfortunately, as continued exponential performance gains through these technologies are no longer possible, the computing industry has embraced multicore as a means of providing ever-increasing peak computing performance. As there is no consensus on multicore architecture, a plethora of efficient, high-performance multicore architectures have emerged. However, the detailed architectural knowledge required to fully exploit these architectures is so prohibitively high that novel programming and optimization tools are required to ensure computational scientists within the DOE Office of Science may reap the benefits of advances in commodity computing technology.

Recently, automatic performance tuning, or auto-tuning, emerged as a technology that provides performance portability of a few key computational kernels from one generation of architecture to the next. Our work on auto-tuning memory-intensive kernels boosted performance by 25x over serial implementations and better than 4x over good parallel implementations, where our optimization and auto-tuning efforts on Cell improved performance by as much as 130x. Our work on graph analytic kernels on GPUs produced speedups of as much as 400x.

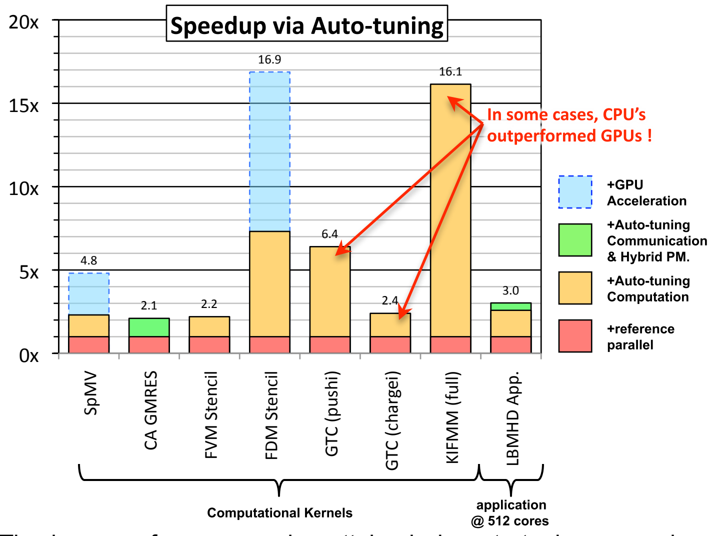
Building on these advances, our proposed research agenda will address auto-tuning's two principal limitations: an interface ill-suited to the forthcoming hybrid SPMD programming model; and its scope limited to fixed-function numerical routines. To that end, we will build a series of broadly-applicable, auto-tuned efficiency layer components. To address auto-tuning's first limitation, we will employ both SPMD ***computational collectives*** and ***concurrent runtimes***. These will hide the complexity of both hybrid communication and multi- and manycore optimization. As such, this model will allow programmers to gracefully transition from the existing flat MPI programming model to a hybrid programming model capable of exploiting the full potential of multicore. Moreover, even if exascale architectures depart dramatically from either of these camps, collectives and runtimes provide a portable abstraction layer that will hide disruptive technological shifts. To address the second limitation, we will develop a runtime for concurrent operations on discrete data structures (deques, sets, and priority queues) and extend the sparse collectives and reduction runtimes to operate on non-numeric data via alternate semirings. Finally, to demonstrate the utility of our method, we will integrate the resultant auto-tuned components into compact applications of interest to the Department of Energy and evaluate them at the largest concurrencies possible.

The development of this technology will play an essential role in the exploitation of multi- and manycore architectures by scientists within the DOE Office of Science over the next decade.

Automatic Performance Tuning

Automatic Performance Tuning (***auto-tuning***) is a proven (ATLAS, FFTW, SPIRAL, OSKI), empirical, feedback-driven, optimization technique designed to address many of the performance optimization challenges seen on multicore processors.

Auto-tuners can be:
- Written for specific numerical methods or applications: **FFTW**, BLAS (**ATLAS, MKL**), SpMV (**OSKI**), CA GMRES, Lattice Boltzmann MHD (LBMHD), Fast Multipole Method (FMM)
- Generated from a Domain Specific Language: **SPIRAL**, stencils
- Generated from parsing C/FORTRAN code.
- Directly incorporated into a compiler.



The large performance gains attained via auto-tuning are only possible by exploiting high-level knowledge about the underlying numerical method. This knowledge allows transformations to data structure and algorithmic parameters that are beyond the reach of compilers. This motivates encapsulation of functionality into large auto-tuned blocks that subsume control of data structure, algorithmic parameters, and possibly parallelism.

Optimizations	Compilers	Auto-tuners
Code/Loop	✔	✔
Data Structures	✗	✔
Parallelism	limited	✔
Synchronization	✗	✔
MPI Communication	✗	✔
Algorithmic Parameters	✗	✔
Implementation Selection	offline, heuristics	runtime, empirical

Auto-tuners can be reused across the breadth and evolution of multicore processors. For example, our SpMV, Stencil, and LBMHD auto-tuners were developed on an older generation of dual-core processors. Nevertheless, they continue to deliver substantial speedups on the latest generation of processors (including Magny Cours, Nehalem-EX, and BlueGene/P).

X-Stack Motivation

Attaining high performance on either a conventional multicore processor (Opteron, BGP, etc…) or novel manycore accelerator (GPU) is often challenging and unproductive as it requires hybrid programming models, detailed architectural knowledge, and a fundamental understanding of the applications in question.

Moreover, we are faced with a conundrum: auto-tuning and threading are often essential for kernel performance, but threading an application can dramatically degrade application performance. Optimizing hybrid applications is as much about accelerating the performance of the threaded routines as it is about minimizing the slowdown (compared to flat MPI) in the non-threaded routines.

Furthermore, the existing set of library-based auto-tuners force programmers to conform to fixed functionality and a fixed interface. This makes integration and broad acceptance challenging. For example, we observe that the same optimizations are applicable in sparse linear algebra regardless of data type (real, complex, integers, or bits) or application domain (PDEs, graph analytics). Similarly, we observe that in the context of particle-mesh methods, although the optimal implementation may be unique to a particular dataset-architecture-application tuple, a generalized optimization design space can capture all of them.

Ultimately, when constructing auto-tuners, we end up solving the same core challenges over and over: management of data movement, data synchronization, and data replication. ***We believe much of this functionality should be extracted, generalized, auto-tuned, encapsulated so that the details are hidden from application scientists, and so that it may be easily reused.***

X-Stack Research Thrusts

- Construction of a series of auto-tuned reduction runtimes that manage data locality, replication, and synchronization at both the intra- and inter-node levels. Such tools will separate the common challenges we've observed in optimizing Particle-in-Cell (PIC) codes from the core underlying method. Moreover, the runtime can be made sufficiently general (alternate semirings) that it can be used in other application fields.

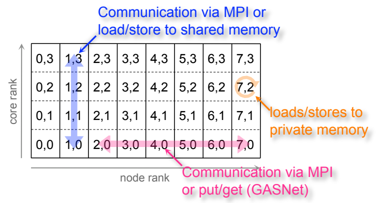
- Construction of a series of auto-tuned routines for discrete data structures (sets, deques, priority queues, etc…) at both the intra- and inter-node levels. Such data structures are increasingly used to implement complex, yet efficient numerical routines. We believe such routines will not only enable further gains in performance on existing applications, but also serve emerging areas of interest to the Office of Science.

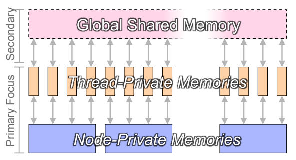
- Integration of the above core routines into generalized (alternate semirings) and auto-tuned sparse linear algebra primitives (SpMV, SpGEMM). Such multicore-optimized tools are clearly critical to scientific applications.

- Integration and evaluation (performance & productivity) of the above into applications in the realms of PIC and Immersed Boundary Method simulations and graph analytics.

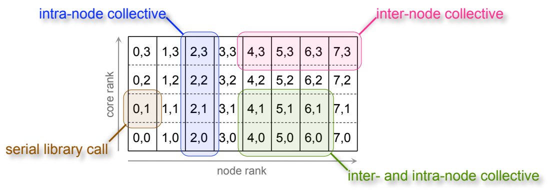
Encapsulation and Abstraction

Directly exploiting shared memory via loads and stores instead of passing messages is essential in attaining high performance and scalability on many applications. Moreover, due to the complexities of their communication patterns, some applications struggle with MPI implementations but are easily expressed in PGAS models. We will adopt a hierarchy of memories and map them to a 2D rank system.

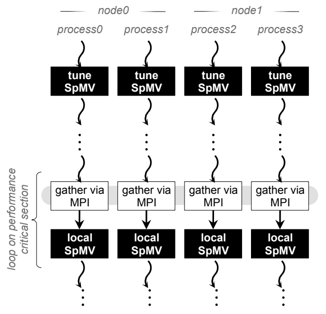

Communication via MPI or load/store to shared memory
loads/stores to private memory
Communication via MPI or put/get (GASNet)
core rank
node rank

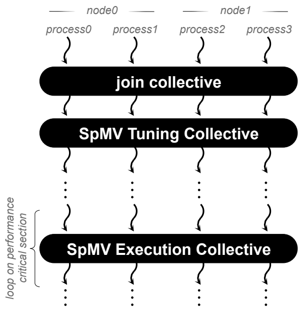


We will extend the well understood concept of communication collectives to perform auto-tuned numerical methods. We differentiate auto-tuned library calls and computational collectives by the degree in which they may collaborate.



Using shared memory extensions, we will construct both a tuning and a computational collective for each primitive to allow black boxing of auto-tuning and easy integration into existing MPI applications.





Interaction with Co-Design and SciDAC Centers

Auto-tuners are a productivity and risk mitigation strategy for both the Co-Design and SciDAC centers. Given the uncertainty in what exascale (or even 100PF) architectures will look like, it is improbable that today's (hand) optimized implementations will run well on such machines. However, an auto-tuner can mitigate this uncertainty by exploring a software design space on each successive generation of processors and adapting to them.

Moreover, as software can adapt to hardware, architects do not need to overprovision their processor designs to suit the needs of legacy applications and implementations. The result is an improvement in power efficiency.

Interaction is key as it ensures our implementations are sufficiently broad and we have the detailed application knowledge to implement the most complex optimizations.